# REPORT DOCUMENTATION PAGE

Form Approved
OPM No.



| 1. AGENCY USE (Leave | 2. REPORT | 3. REPORT TYPE AND DATES |
|---|---|---|
| | | |

4. TITLE AND
CASEWorks/RT Ada i860, Version 1.1, Host: Sun SPARCstation 2
Target: CSPI Supercard 2 (80860) with VSB daughtboard
930722W1.11320

5. FUNDING

AD-A273 713

6.
Authors:

Wright-Patterson AFB

7. PERFORMING ORGANIZATION NAME(S) AND
Ada Validating Facility, Language Control Facility ASD/SCEL
Bldg. 676, Room 135
Wright Patterson AFB, Dayton OH 45433

8. PERFORMING ORGANIZATION

9. SPONSORING/MONITORING AGENCY NAME(S) AND
Ada Joint Program Office
The Pentagon, Rm 3E118
Washington, DC 20301-3080



10. SPONSORING/MONITORING AGENCY

11. SUPPLEMENTARY

12a. DISTRIBUTION/AVAILABILITY
Approved for public release; distribution unlimited

12b. DISTRIBUTION

13. (Maximum 200
[illegible]
CASEWorks/RT Ada i860, Version 1.1, Host: Sun SPARCstation 2, CSPI Supercard 2 (80860) with VSB daughterboard, ACVC 1.11

14. SUBJECT
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility
ANSI/MIL-STD-1815A, AJPO

15. NUMBER OF

16. PRICE

| 17. SECURITY CLASSIFICATION | 18. SECURITY | 19. SECURITY CLASSIFICATION | 20. LIMITATION OF |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED |

NSN

Standard Form 298, (Rev. 2-89)
Prescribed by ANSI Std.

AVF Control Number: AVF-VSR-567.0693
Date VSR Completed: 10 August 1993
93-04-21-MTI

# Ada COMPILER
# VALIDATION SUMMARY REPORT:
Certificate Number: 930722W1.11320
Multiprocessor Toolsmiths Inc.
CASEWorks/RT Ada i860, Version 1.1
Sun SPARCstation 2 under SunOS, 4.1.1 =>
CSPI Supercard 2 (80860) with VSB daughterboard

(Final)

Prepared By:
Ada Validation Facility
645 C-CSG/SCSL
Wright-Patterson AFB OH 45433-5707

## Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on 22 July 1993.

Compiler Name and Version: CASEWorks/RT Ada i860, Version 1.1

Host Computer System: Sun SPARCstation 2
under SunOS, 4.1.1

Target Computer System: CSPI Supercard 2 (80860)
with VSB daughterboard

Customer Agreement Number: 93-04-21-MTI

See section 3.1 for any additional information about the testing environment.

As a result of this validation effort, Validation Certificate 930722W1.11320 is awarded to Multiprocessor Toolsmiths Inc. This certificate expires two years after MIL-STD-1815B is approved by ANSI.

This report has been reviewed and is approved.

Dale E Lange

Ada Validation Facility
Dale E. Lange
Technical Director
645 C-CSG/SCSL
Wright-Patterson AFB OH 45433-5707

for

Ada Validation Organization
Director, Computer and Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond, Director
Department of Defense
Washington DC 20301

DECLARATION OF CONFORMANCE

| | |
|---|---|
| Customer: | Multiprocessor Toolsmiths Inc |
| Ada Validation Facility: | Ada/Jovial Validation Facility<br>ACS/Communications-Computer Systems<br>Wright-Patterson AFB, Ohio 45433-6503 |
| ACVC Version: | 1.11 |
| Ada Implementation: | |
| Compiler Name/ Version: | CASEWorks/RT Ada i860 version 1.1 |
| Host Computer: | Sun SPARCStation 2 |
| Host Operating System: | Sun OS (Unix) 4.1.1 |
| Target Computer: | CSPI Supercard 2 (Intel 80860)<br>with VSB daughterboard |

I, the undersigned, representing Multiprocessor Toolsmiths Inc., declare that Multiprocessor Toolsmiths Inc. has no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A in the implementation listed in this declaration.

Date: 22 July 93

Stephen G. Michell
Ada Program Manager

TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

## 1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

Ada Validation Organization
Computer and Software Engineering Division
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311-1772

## 1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

## 1.3 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable. Class B and class L tests are expected to produce errors at compile time and link time, respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 is used by many tests for Chapter 13 of the Ada Standard. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that all violations of the Ada Standard are detected. Some of the class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in section 2.3.

For each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see section 2.1), and possibly removing some inapplicable tests (see section 2.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

## 1.4 DEFINITION OF TERMS

| | |
|---|---|
| Ada Compiler | The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof. |
| Ada Compiler Validation Capability (ACVC) | The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide and the template for the validation summary report. |
| Ada Implementation | An Ada compiler with its host computer system and its target computer system. |
| Ada Joint Program Office (AJPO) | The part of the certification body which provides policy and guidance for the Ada certification system. |
| Ada Validation Facility (AVF) | The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation. |
| Ada Validation Organization (AVO) | The part of the certification body that provides technical guidance for operations of the Ada certification system. |
| Compliance of an Ada Implementation | The ability of the implementation to pass an ACVC version. |
| Computer System | A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units. |

| | |
|---|---|
| Conformity | Fulfillment by a product, process, or service of all requirements specified. |
| Customer | An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed. |
| Declaration of Conformance | A formal statement from a customer assuring that conformity is realized or attainable on the Ada implementation for which validation status is realized. |
| Host Computer System | A computer system where Ada source programs are transformed into executable form. |
| Inapplicable test | A test that contains one or more test objectives found to be irrelevant for the given Ada implementation. |
| ISO | International Organization for Standardization. |
| LRM | The Ada standard, or Language Reference Manual, published as ANSI/MIL-STD-1815A-1983 and ISO 8652-1987. Citations from the LRM take the form "<section>.<subsection>:<paragraph>." |
| Operating System | Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible. |
| Target Computer System | A computer system where the executable form of Ada programs are executed. |
| Validated Ada Compiler | The compiler of a validated Ada implementation. |
| Validated Ada Implementation | An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro90]. |
| Validation | The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation. |
| Withdrawn test | A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language. |

# CHAPTER 2

# IMPLEMENTATION DEPENDENCIES

## 2.1 WITHDRAWN TESTS

The following tests have been withdrawn by the AVO. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 2 August 1991.

| | | | | | |
|---|---|---|---|---|---|
| E28005C | B28006C | C32203A | C34006D | C35508I | C35508J |
| C35508M | C35508N | C35702A | C35702B | B41308B | C43004A |
| C45114A | C45346A | C45612A | C45612B | C45612C | C45651A |
| C46022A | B49008A | B49008B | A74006A | C74308A | B83022B |
| B83022H | B83025B | B83025D | C83026A | B83026B | C83041A |
| B85001L | C86001F | C94021A | C97116A | C98003B | BA2011A |
| CB7001A | CB7001B | CB7004A | CC1223A | BC1226A | CC1226B |
| BC3009B | BD1B02B | BD1B06A | AD1B08A | BD2A02A | CD2A21E |
| CD2A23E | CD2A32A | CD2A41A | CD2A41E | CD2A87A | CD2B15C |
| BD3006A | BD4008A | CD4022A | CD4022D | CD4024B | CD4024C |
| CD4024D | CD4031A | CD4051D | CD5111A | CD7004C | ED7005D |
| CD7005E | AD7006A | CD7006E | AD7201A | AD7201E | CD7204B |
| AD7206A | BD8002A | BD8004C | CD9005A | CD9005B | CDA201E |
| CE2107I | CE2117A | CE2117B | CE2119B | CE2205B | CE2405A |
| CE3111C | CE3116A | CE3118A | CE3411B | CE3412B | CE3607B |
| CE3607C | CE3607D | CE3812A | CE3814A | CE3902B | |

## 2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

| | |
|---|---|
| C24113L..Y (14 tests) | C35705L..Y (14 tests) |
| C35706L..Y (14 tests) | C35707L..Y (14 tests) |
| C35708L..Y (14 tests) | C35802L..Z (15 tests) |
| C45241L..Y (14 tests) | C45321L..Y (14 tests) |
| C45421L..Y (14 tests) | C45521L..Z (15 tests) |
| C45524L..Z (15 tests) | C45621L..Z (15 tests) |
| C45641L..Y (14 tests) | C46012L..Z (15 tests) |

C35713B, C45423B, B86001T, and C86006H check for the predefined type SHORT_FLOAT; for this implementation, there is no such type.

C35713D and B86001Z check for a predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT; for this implementation, there is no such type.

A35801E includes a check that FLOAT'FIRST..FLOAT'LAST can be used as the range constraint in a floating-point type declaration; this implementation, rejects the declaration. (See section 2.3.)

C45423A, C45523A, and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

D64005G uses 17 levels of recursive procedure calls nesting; this level of nesting for procedure calls exceeds the capacity of the compiler.

B86001Y uses the name of a predefined fixed-point type other than type DURATION; for this implementation, there is no such type.

CA2009C and CA2009F check whether a generic unit can be instantiated before its body (and any of its subunits) is compiled; this implementation creates a dependence on generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic unit bodies makes the instantiating units obsolete. (See section 2.3.)

LA3004A..B, EA3004C..D, and CA3004E..F (6 tests) check pragma INLINE for procedures and functions; this implementation does not support pragma INLINE.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

CD2A84A, CD2A84E, CD2A84I..J (2 tests), and CD2A84O use length clauses to specify non-default sizes for access types; this implementation does not support such sizes.

BD8001A, BD8003A, BD8004A..B (2 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

AE2101C and EE2201D..E (2 tests) use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types with discriminants without defaults; these instantiations are rejected by this compiler.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

| Test | File Operation | Mode | File Access Method |
|---|---|---|---|
| CE2102E | CREATE | OUT_FILE | SEQUENTIAL_IO |
| CE2102F | CREATE | INOUT_FILE | DIRECT_IO |
| CE2102J | CREATE | OUT_FILE | DIRECT_IO |
| CE2102N | OPEN | IN_FILE | SEQUENTIAL_IO |
| CE2102O | RESET | IN_FILE | SEQUENTIAL_IO |
| CE2102R | OPEN | INOUT_FILE | DIRECT_IO |
| CE2102S | RESET | INOUT_FILE | DIRECT_IO |
| CE2102T | OPEN | IN_FILE | DIRECT_IO |
| CE2102U | RESET | IN_FILE | DIRECT_IO |
| CE2102V | OPEN | OUT_FILE | DIRECT_IO |
| CE2102W | RESET | OUT_FILE | DIRECT_IO |
| CE3102F | RESET | Any Mode | TEXT_IO |
| CE3102G | DELETE | ---------- | TEXT_IO |
| CE3102I | CREATE | OUT_FILE | TEXT_IO |
| CE3102J | OPEN | IN_FILE | TEXT_IO |

The tests listed in the following table check the given file operations for the given combination of mode and access method; this implementation does not support these operations.

| Test | File Operation | Mode | File Access Method |
|---|---|---|---|
| CE2105A | CREATE | IN_FILE | SEQUENTIAL_IO |
| CE2105B | CREATE | IN_FILE | DIRECT_IO |
| CE2111A | OPEN | OUT_FILE | SEQUENTIAL_IO |
| CE2208B | OPEN | OUT_FILE | SEQUENTIAL_IO |
| CE3103A | OPEN | OUT_FILE | TEXT_IO |
| CE3104B | OPEN | OUT_FILE | TEXT_IO |
| CE3109A | CREATE | IN_FILE | TEXT_IO |

The following 15 tests check operations on sequential, direct, and text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

| | | | | |
|---|---|---|---|---|
| CE2107B..E | CE2107G..H | CE2107L | CE2110B | CE2110D |
| CE2111H | CE3111B | CE3111D..E | CE3114B | CE3115A |

CE2111C..D (2 tests) reset a SEQUENTIAL_IO file from IN_FILE to OUT_FILE mode; this implementation does not support such a reset.

CE2111F and CE2111I reset a SEQUENTIAL_IO file from OUT_FILE to OUT_FILE mode; this implementation does not support such a reset.

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3104C resets a TEXT_IO file from OUT_FILE to OUT_FILE mode; this implementation does not support such a reset.

CE3104A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

## 2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 9 tests.

The following tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

| | | |
|---|---|---|
| B22003A | B83033B | B85013D |

A35801E was graded inapplicable by Evaluation Modification as directed by the AVO. This test checks that various floating-point attributes can be used in static expressions, and includes the range FLOAT'FIRST..FLOAT'LAST as the constraint in a floating-point type declaration. Such a constraint may be rejected by the compiler if the values for 'FIRST & 'LAST are not within the range of safe numbers for any of the implementation's floating point base types (cf. AI-00469/04). But this implementation rejects the

declaration even though it has a large enough base type; however, this error was not discovered until after testing had begun.

CA2009C and CA2009F were graded inapplicable by Evaluation Modification as directed by the AVO. These tests contain instantiations of a generic unit prior to the compilation of that unit's body; as allowed by AI-00408 and AI-00506, the compilation of the generic unit bodies makes the compilation unit that contains the instantiations obsolete.

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete—no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

CE3104A was graded passed by Test Modification as directed by the AVO. This test checks that the current column, line, and page numbers of text files are set to one after a create, open, or reset operation; it incorrectly allows an execution path wherein an attempt is made to delete a file that is not open, and there is no handler for the resulting STATUS_ERROR. In order to avoid this problem, the test's code was changed as indicated below:

line 198 was changed from ' DELETE (FILE); ' to
' IF IS_OPEN (FILE) THEN DELETE(FILE); END IF; --AVF '

and ' COMMENT ("!!!USE_ERROR: OPEN + OUT_FILE!!!--AVF"); '
was inserted immediately after line 147 (this output confirmed the particular execution path that justified excluding DELETE)

# CHAPTER 3

# PROCESSING INFORMATION

## 3.1 TESTING ENVIRONMENT

The Ada implementation tested in this validation effort is described adequately by the information given in the initial pages of this report.

For technical and sales information about this Ada implementation, contact:

Stephen Michell
Multiprocessor Toolsmiths Inc.
200-6 Gurdwara Drive
Nepean, Ontario
Canada K2E 8A3
(613) 727-8707 Ext:111

Testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

## 3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

The list of items below gives the number of ACVC tests in various categories. All tests were processed, except those that were withdrawn because of test errors (item b; see section 2.1), those that require a floating-point precision that exceeds the implementation's maximum precision (item e; see section 2.2), and those that depend on the support of a file system -- if none is supported (item d). All tests passed, except those that are listed in sections 2.1 and 2.2 (counted in items b

and f, below).

| | | |
|---|---|---|
| a) Total Number of Applicable Tests | 3783 | |
| b) Total Number of Withdrawn Tests | 95 | |
| c) Processed Inapplicable Tests | 91 | |
| d) Non-Processed I/O Tests | 0 | |
| e) Non-Processed Floating-Point Precision Tests | 201 | |
| f) Total Number of Inapplicable Tests | 292 | (c+d+e) |
| g) Total Number of Tests for ACVC 1.11 | 4170 | (a+b+f) |

## 3.3 TEST EXECUTION

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system by the TCP/IP executing over Ethernet, and run. The results were captured on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

| Option/Switch | Effect |
|---|---|
| -fE | Generate the auxiliary file so that the lister will work. |
| -fw | Suppress warnings. |
| -fI | Ignore errors in deciding which units should be added to a library in a given compilation. |
| -fQ | Don't emit messages about adding units to the program library. |
| -L library_name | uses a validation-specific library name |
| -Q "linker_options" | passes through board-specific and build-specific information to Composer(tm) The following Composer selections are used |
| | -b up Use the single-cpu pSOS+ variant of the executive |
| | -r sc80860 Use SC80860-specific modules (timers, serial line drivers) |

| | |
|---|---|
| | for the executive |
| -s bptcpd | Include the backplane TCP/IP server for network communication with the host |
| -s vtty | Include the Virtual Terminal server VTTY for communication with a terminal emulator on the host computer. Used for logging of test results to the host. |
| -s Unix | Include the Unix-compatibility layer. Needed for package Calendar. |
| -s ada | Includes the Ada run-time specific additions to the Unison/pSOS+ executive. |
| -s nfs | Includes the Network File Server to mount a filing system over the network. This server is only included for the applicable CE and EE tests. |
| -s rpc | Includes the Remote Procedure Call server. Only used in these tests with NFS, hence is only included when NFS is included. |

Test output, compiler and linker listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

## APPENDIX A

## MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

| Macro Parameter | Macro Value |
|---|---|
| $MAX_IN_LEN | 200 -- Value of V |
| $BIG_ID1 | (1..V-1 => 'A', V => '1') |
| $BIG_ID2 | (1..V-1 => 'A', V => '2') |
| $BIG_ID3 | (1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A') |
| $BIG_ID4 | (1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A') |
| $BIG_INT_LIT | (1..V-3 => '0') & "298" |
| $BIG_REAL_LIT | (1..V-5 => '0') & "690.0" |
| $BIG_STRING1 | '"' & (1..V/2 => 'A') & '"' |
| $BIG_STRING2 | '"' & (1..V-1-V/2 => 'A') & '1' & '"' |
| $BLANKS | (1..V-20 => ' ') |
| $MAX_LEN_INT_BASED_LITERAL | "2:" & (1..V-5 => '0') & "11:" |
| $MAX_LEN_REAL_BASED_LITERAL | "16:" & (1..V-7 => '0') & "F.E:" |

```
$MAX_STRING_LITERAL    '"' & (1..V-2 => 'A') & '"'
```

The following table lists all of the other macro parameters and their respective values.

| Macro Parameter | Macro Value |
|---|---|
| $ACC_SIZE | 32 |
| $ALIGNMENT | 4 |
| $COUNT_LAST | 2_147_483_646 |
| $DEFAULT_MEM_SIZE | 1024 |
| $DEFAULT_STOR_UNIT | 8 |
| $DEFAULT_SYS_NAME | i860 |
| $DELTA_DOC | 2.0**(-31) |
| $ENTRY_ADDRESS | 16#0# |
| $ENTRY_ADDRESS1 | 16#1# |
| $ENTRY_ADDRESS2 | 16#2# |
| $FIELD_LAST | 2_147_483_647 |
| $FILE_TERMINATOR | ' ' |
| $FIXED_NAME | NO_SUCH_FIXED_TYPE |
| $FLOAT_NAME | NO_SUCH_FLOAT_TYPE |
| $FORM_STRING | "" |
| $FORM_STRING2 | "CANNOT_RESTRICT_FILE_CAPACITY" |
| $GREATER_THAN_DURATION | 90_000.0 |
| $GREATER_THAN_DURATION_BASE_LAST | 10_000_000.0 |
| $GREATER_THAN_FLOAT_BASE_LAST | 1.8E+308 |
| $GREATER_THAN_FLOAT_SAFE_LARGE | 3.0E38 |

$GREATER_THAN_SHORT_FLOAT_SAFE_LARGE
1.0E308

$HIGH_PRIORITY 250

$ILLEGAL_EXTERNAL_FILE_NAME1
/NODIRECTORY/FILENAME1

$ILLEGAL_EXTERNAL_FILE_NAME2
/NODIRECTORY/FILENAME2

$INAPPROPRIATE_LINE_LENGTH
-1

$INAPPROPRIATE_PAGE_LENGTH
-1

$INCLUDE_PRAGMA1 PRAGMA INCLUDE ("A28006D1.ADA")

$INCLUDE_PRAGMA2 PRAGMA INCLUDE ("B28006F1.ADA")

$INTEGER_FIRST -2147483648

$INTEGER_LAST 2147483647

$INTEGER_LAST_PLUS_1 2_147_483_648

$INTERFACE_LANGUAGE C

$LESS_THAN_DURATION -90_000.0

$LESS_THAN_DURATION_BASE_FIRST
-10_000_000.0

$LINE_TERMINATOR ASCII.LF

$LOW_PRIORITY 1

$MACHINE_CODE_STATEMENT
NULL;

$MACHINE_CODE_TYPE INSTRUCTION

$MANTISSA_DOC 31

$MAX_DIGITS 15

$MAX_INT 2147483647

$MAX_INT_PLUS_1 2_147_483_648

$MIN_INT -2147483648

$NAME BYTE_INTEGER

MACRO PARAMETERS

| | |
|---|---|
| $NAME_LIST | i860 |
| $NAME_SPECIFICATION1 | /home/sparc6/aval/val.SPARC/X2120A |
| $NAME_SPECIFICATION2 | /home/sparc6/aval/val.SPARC/X2120B |
| $NAME_SPECIFICATION3 | /home/sparc6/aval/val.SPARC/X3119A |
| $NEG_BASED_INT | 16#FFFFFFFE# |
| $NEW_MEM_SIZE | 1024 |
| $NEW_STOR_UNIT | 8 |
| $NEW_SYS_NAME | i860 |
| $PAGE_TERMINATOR | ASCII.LF & ASCII.FF |
| $RECORD_DEFINITION | NEW INTEGER |
| $RECORD_NAME | INSTRUCTION |
| $TASK_SIZE | 32 |
| $TASK_STORAGE_SIZE | 8000 |
| $TICK | 0.025 |
| $VARIABLE_ADDRESS | FCNDECL.VAR_ADDRESS |
| $VARIABLE_ADDRESS1 | FCNDECL.VAR_ADDRESS1 |
| $VARIABLE_ADDRESS2 | FCNDECL.VAR_ADDRESS2 |
| $YOUR_PRAGMA | NO_SUCH_PRAGMA |

## APPENDIX B

## COMPILATION SYSTEM OPTIONS

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report.

-L - Perform compile on a program library named something other than ada.lib.

-g - Run additional optimization pass.

-K - Keep internal form file. This option is used in conjunction with the Optimizer.

-k - Save all of the intermediate files. This option is useful for viewing or debugging the intermediate "C" code generated by the back end of the compiler.

-l " modifiers" -
Generate listing file. The -l option causes the compiler to create a listing. Optional modifiers may be given to affect the listing format. Any combination of the following modifiers can be used:

- c - continuous listing format.
- p - obey pragma page directives.
- s - use standard output.
- t - relevant text output only.

-fv - Verbose mode. Print out the names of the units with'd, and what units are added to the symbol table.

-fD - Debug mode. Generate debugging information.

-fq - Quiet mode. Don't print any information about a compilation.

-fL - Enables the compiler to give the filename and line numbers where an exception takes place.

-fr - Do not release memory at compile time.

-fU - Compile this unit, but do not update the program library.

-fw - Suppress warnings.

-fs - Compile time switch to suppress checking operations.

-fN - Compile time switch to suppress numeric checking.

-fz - Emit code to zero all locals.

-fI - Ignore errors in deciding which units should be added to a library in a given compilation (see ea1003b).

-fE - Generate the auxiliary file so that the lister will work.

-fR - Generate ROMable code.

-fQ - Don't emit messages about adding units to the program library.

-fo - Don't update the program library if a change is made that won't affect the program library (useful when spec and body are in the same file).

-fP - Change the .int extension to .pre. This is used when the optimizer is invoked.

-fC - Compile only if this unit is out-of-date. This is used in conjunction with the tool "adaorder".

-fV - Change common to data. Explicitly initialize globals so that objects that would go into the common area will be placed in the data area.

-fT - Trace C functions. This inserts tracing code on entry and exit from each subprogram. This is useful on a system with no debugger when it is first being brought up.

-fb - Debug mode. Output iform blocks to a .b file.

-fe - Intersperse source lines as comments in C (assembly) code.

-fn - Output translations of globals to C names to .n file.

## LINKER OPTIONS

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this appendix are to linker documentation and not to this report.

-c - Use a different compiler than the one specified for this

program library.

-L - Perform bamp on a program library named something other than ada.lib.

-o - Takes a parameter. This specifies a different name for the executable.

-r - Produce a relocatable object.

-s - Specify an explicit default for the size of a task stack.

-N - Don't actually call the linker. Prints out the program steps which would be executed in the bamp phase

-P - Print the steps (or subtools) invoked as part of bamp.

-v - Verbose mode.

-q - Quiet mode.

-k - Save all of the intermediate files associated with bamp.

-f - Don't compile the elaboration main routine.

-n - Stop before the link step.

-m - Create a link map.

-u - Target software floating point.

-I - Link in Preemptive tasking runtime.

-g; -G - Invoke the global optimizer.

-z - Create executable that will perform dynamic linking.

-Q "linker_options" passes through board-specific and build-specific information to Composer(tm) The following Composer selections may be used in this implementation:

| | |
|---|---|
| -b up | Use the single-cpu pSOS+ variant of the executive |
| -r sc80860 | Use SC80860-specific modules (timers, serial line drivers) for the executive (other options are sc3 and sc6 |
| -s bptcpd | Include the backplane TCP/IP server for network communication with the host |
| -s vtty | Include the Virtual Terminal server VTTY for communication |

| | |
|---|---|
| | with a terminal emulator on the host computer. Used for logging of test results to the host. |
| -s unix | Include the Unix-compatibility layer. Needed for package Calendar. |
| -s ada | Includes the Ada run-time specific additions to the Unison/pSOS+ executive. |
| -s nfs | Includes the Network File Server to mount a filing system over the network. This server is only included for the applicable CE and EE tests. |
| -s rpc | Includes the Remote Procedure Call server. Only used in these tests with NFS, hence is only included when NFS is included. |

# APPENDIX C

## APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is
   ..........
   type INTEGER is range -2147483648 .. 2147483647;
   type LONG_INTEGER is range -2147483648 .. 2147483647;
   type SHORT_INTEGER is range -32768 .. 32767;
   type BYTE_INTEGER is range -128 .. 127;

   type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;
   type LONG_FLOAT is
         digits 15 range -1.79769313486231E+308 .. 1.79769313486231E+308;

   type DURATION is delta 0.0001 range -86400.0000 ..86400.0000;
   ..........
end STANDARD;
```

# Appendix F
# Implementation-Dependent Characteristics

This appendix lists implementation-dependent characteristics of CASEWorks/RT Ada for the Intel 80860 executing with Unison/pSOS+ in an embedded environment. Note that there are no preceding appendices. This appendix is called Appendix F in order to comply with the Reference Manual for the Ada Programming Language* (LRM) ANSI/MIL-STD-1815A which states that this appendix be named Appendix F.

Implemented Chapter 13 features include length clauses, enumeration representation clauses, record representation clauses, address clauses, interrupts, package system, pragma interface, and unchecked programming.

## F.1 Pragmas

The implemented pre-defined pragmas are:

| | |
|---|---|
| elaborate | See the LRM section 10.5 |
| interface | See section F.1.1 |
| list | See the LRM Appendix B |
| pack | See section F.1.2 |
| page | See the LRM Appendix B |
| priority | See the LRM Appendix B |
| suppress | See section F.1.3 |
| shared | See the LRM, chapter 13. |

The remaining pre-defined pragmas are accepted, but presently ignored:

- controlled
- optimize
- inline
- system_name
- storage_unit
- memory_size

Named parameter notation for pragmas is not supported.
When illegal parameter forms are encountered at compile time, the compiler issues a warning message rather than an error, as required by the Ada language definition. Refer to the LRM Appendix B for additional information about the pre-defined pragmas.

### F.1.1 Pragma Interface

The form of pragma interface in CASEWorks/RT Ada is:

```
pragma interface( language, subprogram [,"link-name"] );
```

where:

language This is the interface language, one of the names assembly, builtin, c, or internal. The names builtin and internal are reserved for use by Toolsmiths compiler maintainers in run-time support packages.

subprogram
This is the name of a subprogram to which the pragma interface applies.

link-name This is an optional string literal specifying the name of the non-Ada subprogram corresponding to the Ada subprogram named in the second parameter. If link-name is omitted, then link-name defaults to the value of subprogram translated to lowercase. Depending on the language specified, some automatic modifications may be made to the link-name to produce the actual object code symbol name that is generated whenever references are made to the corresponding Ada subprogram. The object code symbol generated for link-name is always translated to upper case.

It is appropriate to use the optional link-name parameter to pragma interface only when the interface subprogram has a name that does not correspond at all to its Ada identifier or when the interface subprogram name cannot be given using rules for constructing Ada identifiers (e.g., if the name contains a '$' character).

The characteristics of object code symbols generated for each interface language are:

assembly - The object code symbol is the same as link-name.

builtin - The object code symbol is the same as link-name, but prefixed with the following characters (_mss_). This language interface is reserved for special interfaces defined by Toolsmiths. The builtin interface is presently used to declare certain low-level run-time operations whose names must not conflict with programmer-defined or language system defined names.

c - The object code symbol is othe same as link-name, but with one underscore character (_) prepended. This is the convention used by the C compiler.

internal - No object code symbol is generated for an internal language interface; this language interface is reserved for special interfaces defined by Toolsmiths. The internal interface is presently used to declare certain machine-level bit operations.

No automatic data conversions are performed on parameters of any interface subprograms. It is up to the programmer to ensure that calling conventions match and that any necessary data conversions take place when calling interface subprograms.
A pragma interface may appear within the same declarative part as the subprogram to which the pragma interface applies, following the subprogram declaration, and prior to the first use of the subprogram. A pragma interface that applies to a subprogram declared in a package specification must occur within the same package specification as the subprogram declaration; the pragma interface may not appear in the package body in this case. A pragma interface declaration for either a private or nonprivate subprogram declaration may appear in the private part of a package specification.
Pragma interface for library units is not supported.
Refer to the LRM section 13.9 for additional information about pragma interface.

### F.1.2 Pragma Pack

Pragma pack is implemented for composite types (records and arrays).

Pragma pack is permitted following the composite type declaration to which it applies, provided that the pragma occurs within the same declarative part as the composite type declaration, before any objects or components of the composite type are declared.
Note that the declarative part restriction means that the type declaration and accompanying pragma pack cannot be split across a package specification and body.
The effect of pragma pack is to minimize storage consumption by discrete component types whose ranges permit packing. Use of pragma pack does not defeat allocations of alignment storage gaps for some record types. Pragma pack does not affect the representations of real types, pre-defined integer types, and access types.

### F.1.3 Pragma Suppress

Pragma suppress is implemented as described in the LRM section 11.7, with these differences:

Presently, division_check and overflow_check must be

suppressed via a compiler flag, -fN; pragma suppress is ignored for these two numeric checks.

The optional ON =>> parameter name notation for pragma suppress is ignored.

The optional second parameter to pragma suppress is ignored; the pragma always applies to the entire scope in which it appears.

### F.1.4 Pragma Shared

Pragma shared tells the compiler that the variable specified by this pragma is to be shared between 2 or more tasks and that any update or read of this variable should be synchronized with all other tasks which have visibility to the data item. CASEWorks/RT Ada's approach to shared variables is to have the compiler generate calls to a user-accessible function for each read or update of to which a pragma shared has been applied. These functions have been implemented by CASEWorks/Ada and pragma shared is supported. Should a project need extra behaviour, such as disabling interrupts, flushing cache, or setting a semaphore before each update, these subprograms can be rewritten and recompiled into a project library, so long as the specifications exactly match those specifications which follow.

```
with system;
with access_types;
with unsigned_types;

package pragma_shared is

  -- shared get routines --

 function  shared_get_byte (
              byte_p : in  access_byte_integer)
   return byte_integer;

 function  shared_get_unsigned_byte (
       unsigned_byte_p : in access_unsigned_byte_integer)
   return unsigned_types.unsigned_byte_integer;

 function  shared_get_int (
              integer_p : in  access_integer)
   return integer;

 function  shared_get_pointer (
              integer_p : in  access_integer)
   return system.address;

 -- shared assigns --

 procedure shared_asgn_byte (
```

```
                byte_p : in     access_byte_integer;
                value  : in     byte_integer);


    procedure shared_asgn_unsigned_byte (
                byte_p : in access_unsigned_byte_integer;
                value  : in unsigned_byte_integer);

    procedure shared_asgn_int (
                integer_p : in     access_integer;
                value     : in     integer);

    procedure shared_asgn_pointer (
                integer_p : in access_integer;
                value     : in system.address);

  end pragma_shared;
```

F.1.5 Implementation-Defined Pragmas

The following Implementation-defined pragmas are defined by CASEWorks/RT Ada.

Pragma Volatile

F.1.7 Pragma Volatile

This pragma applies to the same Ada types as Pragma Shared - simple types and access types. Pragma Volatile applied to a variable directs the Ada compiler to never store this variable as a register variable but to always perform all reads and writes to memory.

F.2 Attributes

All attributes described in the LRM Appendix A are supported.

F.3 Standard Types

Additional standard types are defined in CASEWorks/RT Ada:

- byte_integer
- short_integer
- long_integer
- long_float

The standard numeric types are defined as:

```
    type byte_integer  is range -128 .. 127;

    type short_integer is range -32768  ..  32767;
```

```
type integer       is range -2147483648  ..  2147483647;

type long_integer  is range -2147483648  ..  2147483647;

type float         is digits 6
                   range -3.40282E+38  ..  3.40282E+38;

type long_float    is digits 15
                   range -1.79769313486231E+308
                   ..  1.79769313486231E+308;

type duration      is delta 0.0001
                   range -86400.0000  ..  86400.0000;
```

## F.4 Package System

The specification of package system is:

```
package system is
  type address is new long_integer;

  type name is (i860);
  system_name   : constant name := i860;

  storage_unit  : constant := 8;
  memory_size   : constant := 1024;

  -- System-Dependent Named Numbers

  min_int       : constant := -2147483648;
  max_int       : constant := 2147483647;
  max_digits    : constant := 15;
  max_mantissa  : constant := 31;
  fine_delta    : constant := 2.0 ** (-31);
  tick          : constant := 20.0/1000.0;

  -- Other System-Dependent Declarations

  subtype priority is integer range 1 .. 250;
```

The value of system.memory_size is presently meaningless.

## F.5 Restrictions on Representation Clauses

### F.5.1 Length Clauses

A size specification (t'size) is rejected if fewer bits are specified than can accommodate the type. The minimum size of a composite type may be subject to application of pragma pack. It is permitted to specify precise sizes for unsigned integer ranges,

e.g., 8 for the range 0..255. However, because of requirements imposed by the Ada language definition, a full 32-bit range of unsigned values, i.e. 0..(2**32)-1, cannot be defined, even using a size specification.

The specification of collection size (t'storage_size) is evaluated at run-time when the scope of the type to which the length clause applies is entered, and is therefore subject to rejection (via storage_error) based on available storage at the time the allocation is made. A collection may include storage used for run-time administration of the collection, and therefore should not be expected to accommodate a specific number of objects. Furthermore, certain classes of objects such as unconstrained discriminant array components of records may be allocated outside a given collection, so a collection may accommodate more objects than might be expected.

The specification of storage for a task activation (t'storage_size) is evaluated at run-time when a task to which the length clause applies is activated, and is therefore subject to rejection (via storage_error) based on available storage at the time the allocation is made. Storage reserved for a task activation is separate from storage needed for any collections defined within a task body.

The specification of small for a fixed point type (t'small) is subject only to restrictions defined in the LRM section 13.2.

### F.5.2 Enumeration Representation Clauses

The internal code for the literal of an enumeration type named in an enumeration representation clause must be in the range of standard.integer.
The value of an internal code may be obtained by applying an appropriate instantiation of unchecked_conversion to an integer type.

### F.5.3 Record Representation Clauses

The storage unit offset (the "at static_simple_expression" part) is given in terms of 8-bit storage units and must be even.
A bit position (the range part) applied to a discrete type component may be in the range 0..15, with 0 being the least significant bit of a component. A range specification may not specify a size smaller than can accommodate the component. A range specification for a component not accommodating bit packing may have a higher upper bound as appropriate (e.g., 0..31 for a discriminant string component). Refer to the internal data representation of a given component in determining the component size and assigning offsets.

Components of discrete types for which bit positions are specified may not straddle 16-bit word boundaries, unless they begin at bit position 0. In these cases the compiler will allocate 32 bits for the unit, and will not permit other components to be assigned to these bits.

The value of an alignment clause (the optional at mod part) must evaluate to 1, 2, 4, or 8, and may not be smaller than the highest alignment required by any component of the record. This means that some records may not have alignment clauses smaller than 2.

### F.5.4 Address Clauses

An address clause may be supplied for an object (whether constant or variable) or a task entry, but not for a subprogram, package, or task unit. The meaning of an address clause supplied for a task entry is given in section F.5.5>.

An address expression for an object is a 32-bit segmented memory address of type system.address.

### F 5.5 Interrupts

A task entry's address clause can be used to associate the entry with a physical hardware address. Values supplied are hardware dependant. See the CASEWorks/RT reference manual and the processor reference manual for interrupt ranges on your processor.
An interrupt entry may not have any parameters.

### F.5.6 Change of Representation

There are no restrictions for changes of representation effected by means of type conversion.

## F.6 Implementation-Dependent Components

No names are generated by the implementation to denote implementation-dependent components.

## F.7 Unchecked Conversions

There are no restrictions on the use of unchecked conversion. Conversions between objects whose sizes or whose basic machine representation do not conform may result in

Storage areas with undefined values.

Wrong values read because of differing underlying representations, such as a character stored in the high-order

byte of a 32-bit integer.

Exceptions due to misaligned data, trying to read an integer on a 2-byte boundary on a RISC processor, for example.

## F.8 Input-Output Packages

A summary of the implementation-dependent input-output characteristics is:

In calls to open and create, the "form" parameter must be the empty string (the default value).

More than one internal file can be associated with a single external file for reading only. For writing, only one internal file may be associated with an external file; do not use reset to get around this rule since

- Reset from IN_FILE mode to OUT_FILE mode will raise USE_ERROR for Sequential_IO and Text_IO opened files
- Direct_IO files are internally buffered and output to the physical file will be different and possibly unpredictable or corrupted.

Temporary sequential and direct files are given names. Temporary files are deleted when they are closed.

In calls to reset, out_file and inout_files cannot be reset, except as IN_FILE. This is because the underlying NFS must delete and attempt to recreate the file. Network timeout issues preclude this activity currently.
Files cannot be deleted and recreated with the same name until the network timeout has expired - about 4 minutes. This limitation applies to deleting a file and attempting to recreate it within the same program, or attempting to re-execute the same program within this time.

File I/O is buffered; text files associated with terminal devices are line-buffered.

The packages sequential_io and direct_io cannot be instantiated with unconstrained composite types or record types with discriminants without defaults.

## F.9 Source Line and Identifier Lengths

Source lines and identifiers in Ada source programs are presently limited to 200 characters in length.